

Topic 13

procedural design and Strings

“Ugly programs are like ugly suspension bridges: they're much more liable to collapse than pretty ones, because the way humans (especially engineer-humans) perceive beauty is intimately related to our ability to process and understand complexity.”

- Eric S. Raymond,

Author of *The Cathedral and the Bazaar*

Copyright Pearson Education, 2010
Based on slides by Marty Stepp and Stuart Reges
from <http://www.buildingjavaprograms.com/>

Nested if/else question

Formula for body mass index (BMI):

$$BMI = \frac{weight}{height^2} \times 703$$

- Write a program that produces output like the following:

This program reads data for two people and computes their body mass index (BMI) and weight status.

Enter next person's information:
height (in inches)? 73.5
weight (in pounds)? 230
BMI = 29.93
overweight

Enter next person's information:
height (in inches)? 71
weight (in pounds)? 220.5
BMI = 30.75
obese

Difference = 0.82

BMI	Weight class
below 18.5	underweight
18.5 - 24.9	normal
25.0 - 29.9	overweight
30.0 and up	obese

2

One-person, no methods

```
import java.util.*;

public class BMI {
    public static void main(String[] args) {
        System.out.println("This program reads ... (etc.)");
        Scanner console = new Scanner(System.in);

        System.out.println("Enter next person's information:");
        System.out.print("height (in inches)? ");
        double height = console.nextDouble();

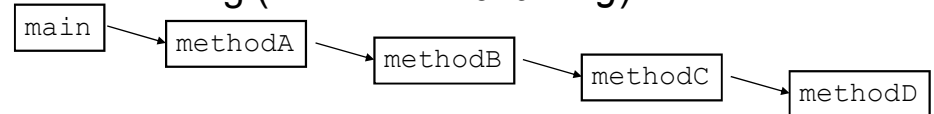
        System.out.print("weight (in pounds)? ");
        double weight = console.nextDouble();

        double bmi = weight * 703 / height / height;
        System.out.printf("BMI = %.2f\n", bmi);
        if (bmi < 18.5) {
            System.out.println("underweight");
        } else if (bmi < 25) {
            System.out.println("normal");
        } else if (bmi < 30) {
            System.out.println("overweight");
        } else {
            System.out.println("obese");
        }
    }
}
```

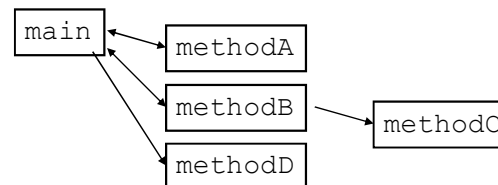
3

"Chaining"

- main should be a concise summary of your program.
 - It is bad if each method calls the next without ever returning (we call this *chaining*):



- A better structure has main make most of the calls.
 - Methods must return values to main to be passed on later.



4

Bad "chain" code

```
public class BMI {
    public static void main(String[] args) {
        System.out.println("This program reads ... (etc.)");
        Scanner console = new Scanner(System.in);
        person(console);
    }

    public static void person(Scanner console) {
        System.out.println("Enter next person's information:");
        System.out.print("height (in inches)? ");
        double height = console.nextDouble();
        getWeight(console, height);
    }

    public static void getWeight(Scanner console, double height) {
        System.out.print("weight (in pounds)? ");
        double weight = console.nextDouble();
        computeBMI(console, height, weight);
    }

    public static void computeBMI(Scanner s, double h, double w) {
        ...
    }
}
```

5

Procedural heuristics

1. Each method should have a clear responsibility.
2. No method should do too large a share of the overall task.
3. Minimize coupling and dependencies between methods.
4. The main method should read as a concise summary of the overall set of tasks performed by the program.
5. Variables should be declared/used at the lowest level possible.

6

Better solution

```
// This program computes two people's body mass index (BMI) and
// compares them.
// The code uses Scanner for input, and parameters/returns.

import java.util.Scanner;

public class BMI {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);
        double bmi1 = person(console);
        double bmi2 = person(console);

        // report overall results
        report(1, bmi1);
        report(2, bmi2);
        System.out.println("Difference = "
                           + Math.abs(bmi1 - bmi2));
    }

    // prints a welcome message explaining the program
    public static void introduction() {
        System.out.println("This program reads ...");
        // ...
    }
}
```

7

Better solution, cont'd.

```
// reads information for one person, computes their BMI, and returns it
public static double person(Scanner console) {
    System.out.println("Enter next person's information:");
    System.out.print("height (in inches)? ");
    double height = console.nextDouble();

    System.out.print("weight (in pounds)? ");
    double weight = console.nextDouble();
    System.out.println();

    return bmi(height, weight);
}

// Computes/returns a person's BMI based on their height and weight.
public static double bmi(double height, double weight) {
    return weight * 703 / (height * height);
}

// Outputs information about a person's BMI and weight status.
public static void report(int number, double bmi) {
    System.out.printf("Subject%5dBMI = %.2f\n", number, bmi);
    if (bmi < 18.5) {
        System.out.println("underweight");
    } else if (bmi < 25) {
        System.out.println("normal");
    } else if (bmi < 30) {
        System.out.println("overweight");
    } else {
        System.out.println("obese");
    }
}
}
```

8

Strings

- **string**: An object storing a sequence of text characters.
 - Unlike most other objects, a `String` is not always created with `new`.

```
String name = "text";  
String name = expression;
```

- Examples:

```
String name = "Marla Singer";  
int x = 3;  
int y = 5;  
String point = "(" + x + ", " + y + ")"; 9
```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String name = "K. Scott";
```

index	0	1	2	3	4	5	6	7
character	K	.		S	c	o	t	t

- First character's index : 0 (zero based indexing)
- Last character's index : 1 less than the string's length
- The individual characters are values of type `char` (another primitive data type)

10

String methods

Method name	Description
<code>indexOf(str)</code>	index where the start of the given string appears in this string (-1 if not found)
<code>length()</code>	number of characters in this string
<code>substring(index1, index2)</code> or <code>substring(index1)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (exclusive); if <i>index2</i> is omitted, grabs till end of string
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String student = "Olivia Scott";  
System.out.println(student.length()); // 12
```

11

String method examples

```
// index    012345678901  
String s1 = "Olivia Scott";  
String s2 = "Isabelle Scott";  
System.out.println(s2.length()); // 14  
System.out.println(s1.indexOf("e")); // -1  
System.out.println(s2.indexOf("e")); // 4  
System.out.println(s1.substring(7, 10)); // "Sco"  
String s3 = s2.substring(4, 10);  
System.out.println(s3.toLowerCase()); // "elle s"
```

- Given the following string:

```
// index    0123456789012345678901  
String book = "Building Java Programs";
```

- How would you extract the word "Building" ?
(Write code that can extract the first word from any string.)

12

Clicker 1

- What is output by the following code?

```
String s1 = "Football";  
String s2 = s1.substring(4, 8);  
s2.substring(1);  
System.out.print(s2);
```

- A. Football
- B. ball
- C. all
- D. No output due to syntax error.
- E. No output due to runtime error.

13

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "ut Longhorns";  
s.toUpperCase();  
System.out.println(s);    // ut Longhorns
```

- To modify a variable's value, you must reassign it:

```
String s = "ut Longhorns";  
s = s.toUpperCase();  
System.out.println(s);    // UT LONGHORNS
```

14

Strings as user input

- Scanner's `nextLine` method reads a word of input as a String.

```
Scanner console = new Scanner(System.in);  
System.out.print("What is your first name? ");  
String name = console.nextLine();  
System.out.println(name + " has " + name.length() +  
    " letters and starts with " + name.substring(0, 1));
```

Output:

```
What is your first name? Chamillionaire  
Chamillionaire has 14 letters and starts with C
```

- The `nextLine` method reads a line of input as a String.

```
System.out.print("What is your address? ");  
String address = console.nextLine();
```

15

Clicker 2

- What is output by the following code?

```
String s1 = "taxicab";  
String s2 = "acables";  
String s3 = s1.substring(4);  
String s4 = s2.substring(1, 4);  
if (s3.length() == s4.length())  
    System.out.print("1");  
else  
    System.out.print("2");  
if (s3 == s4)  
    System.out.print("1");  
else  
    System.out.print("2");
```

- A. 11
- B. 12
- C. 21
- D. 22
- E. No output due to syntax error

16

Comparing Strings

- Relational operators such as < and <= are undefined on objects in Java.
- == is defined but normally doesn't work as intended

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

- This code will compile, but it will not print the song.
- == compares objects by *references* (seen later), so it often gives false even when two Strings have the same letters.

17

The equals method

- Objects are compared using a method named equals.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("Fred's Friend.");
    System.out.println("Purple Dinasuar.");
    System.out.println("In trouble.");
}
```

- The equals method returns a value of type boolean, the type used in logical tests.

18

String test methods

Method	Description
equals(str)	whether two strings contain the same characters
equalsIgnoreCase(str)	whether two strings contain the same characters, ignoring upper vs. lower case
startsWith(str)	whether one contains other's characters at start
endsWith(str)	whether one contains other's characters at end
contains(str)	whether the given string is found within this one

```
String name = console.next();
if (name.startsWith("Prof")) {
    System.out.println("When are your office hours?");
} else if (name.endsWith("OBE")) {
    System.out.println("Yes Sir!");
}
```

19

Strings questions

- Write a method to determine if a String is a possible representation of a DNA strand
 - contains only A, C, T, and G
- Write a method to create a *Watson-Crick complement* given a String that represents a strand of DNA
 - replace A with T, C with G, and vice versa
- Given a String that represents a strand of DNA return the first substring that exists between "ATG" and either "TAG" or "TGA"
 - no overlap allowed

20

String Questions

- Write a method that returns the number of times a given character occurs in a String
- Write a method that returns the number of times the punctuation marks . ? ! , : " ; ' occur in a String